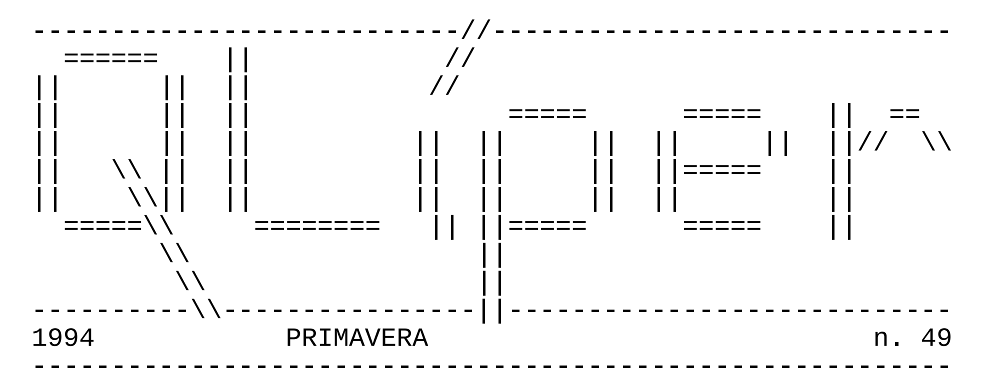


QLíper
Redactor: Salvador Merino
Tel. +34-(9)5-2475043
Cerámicas Mary
Ctra. Cádiz (Torreblanca del Sol)
ES-29640 FUENGIROLA

Cuota Socio

La cuota socio es anual (Comienza en Enero y termina en Diciembre).
La salida de QLíper es TRIMESTRAL (4 discos al año).

ESPAÑA............ 1500 Pesetas.
EUROPA (CE)....... 1500 Pesetas.
RESTO MUNDO ...... 2000 Pesetas o US $15

Editorial

En el momento de escribir esta Editorial, solamente han renovado 10 socios, pero aún estoy convencido de que en los próximos días van a renovar un minimo de 3 socios más.

La situación económica de QLíper es muy buena. No solamente tenemos dinero para cubrir todos los gastos del Club durante 1994, sino que podemos permitirnos el lujo de realizar un extra verano o navidad. También podemos realizar una mini campaña de publicidad para recuperar algún viejo socio, pero dado que sus direcciones son muy antiguas (6 años sin contactar con ellos), no tengo muchas esperanzas de que aún vivan en ese domicilio, y menos de que estén interesados en hacerse socios de un Club dedicado a un ordenador olvidado con el paso del tiempo.

Este QLíper ha sido elaborado y enviado antes de su fecha para demostrar que aunque la mayoria de los socios se niegan a colaborar en la compilación, vuestro Editor (o "Capo máximo de QLíper") es capaz de realizar trucos de magia como compilar 3 QLiperes en 28 días.

La compilación del próximo QLíper será durante mediados de primavera. A ver si me enviais algo, y no tengo que echar mano a los intercambios Internacionales.

Salvador Merino, 26/1/1994

EN ESTE DISCO...

================

```
QLIPER49_SCR       (portada disco QLíper)
inventario_txt     (Tesoro de QLíper)
oferta_doc         (La programoteca QLíper)
QLAGENCY26_TXT     (Noticias Internacionales de QITALY 26).
- VerPanQL         (Pedro Reina)
- cndate           (Pedro Reina)
- OLIMPO 1.1       (Pedro Reina)
- MERINO TIL v2.0  (Salvador Merino)
```

Cuando estaba programando Olimpo 1.1 me encontré con que tenía que leer con C68 la fecha del reloj interno del QL y después convertirlo a un formato legible.

Como sabéis, el reloj del QL lo que cuenta es el número de segundos desde las 00:00:00 del 1 de enero de 1961.

C68 cuenta con la función mt_rclck() para obtener ese número (de 4 octetos); pero el "problema" lo tuve para traducir ese numerajo a fecha, algo que SuperBASIC hace sencillamente en la función DATE$. El QDOS tiene una utilidad por vector llamada CN.DATE que resuelve el problema, pero la versión 3.05 de C68 no permite usarla directamente. Afortunadamente, la 4.03 ya sí lo permite, de modo que perfecto...

Problema: la función de C68 cn_date() efectivamente devuelve la fecha del reloj, pero de una manera muy peculiar, que no se puede usar directamente en C.

Para que podáis usarla sin problemas os he escrito el miniprograma cndate_c en el que leo el reloj y obtengo el día, mes y año del sistema. Podéis ver cómo funciona tecleando

EXEC FLP1_cndate_exe

aunque, claro lo interesante es ver el código, ya que esto sólo os interesa a los que deseáis programar con C68. ¡Feliz código!

Pedro Reina, J.13.1.1994

COMENTARIOS OLIMPO
=====================

Olimpo sigue su marcha y ahora aparece la versión 1.1, con algunas novedades: por fín empieza a manejar gráficos, ya puede leer bases de datos en formato compatible con dBase y maneja listas encadenadas. Por lo demás, he pulido algunos detalles.

Para ver la demostración debéis tener la pantalla en modo 4 y teclear

EXEC_W Demo_exe

el programa utiliza el directorio para datos de Toolkit 2, de modo que si arrancáis el programa desde FLP1_, por ejemplo, para que pueda encontrar los ficheros que busca en algún momento teníais que haber escrito en algún momento

DATA_USE FLP1_

Si no tenéis Toolkit 2 el programa también funciona, pero algunas cosas no las podréis ver.

Para que comprobéis el manejo de base de datos os recomiendo que hagáis lo siguiente:

Elegir una base de datos en un PC que tenga formato compatible dBase y sin campos memo; por ejemplo, tenéis por ahí el fichero AGENDA.DBF.

Lo traéis al QL con cualquier programa de transferencia de modo que no cambie ningún carácter, es decir, tratado como fichero binario. Esto os dará el fichero AGENDA_DBF

En el programa de demostración, usad la opción "Base dat" y cuando os lo pida, escribid AGENDA_DBF; el programa os mostrará la estructura de la base de datos, diciendo cómo se definen los primeros campos.

Como Olimpo es compatible entre QL y PC, el programa de demostración Demo_c se puede compilar en un PC. Para que comprobéis que la cosa funciona bien, llevaos el fichero Demo.exe a un PC y ejecutarlo. Si tenéis el QL y el PC al lado veréis que los programas funcionan igual. Debéis llevaros también los ficheros Demo_cnf (como fichero de texto) y Demo_dbf (como binario).

Recordad que Olimpo es de dominio público. Está en la librería PD de QLíper, pero es mejor que me lo pidáis a mí si os interesa, ya que os mandaré el manual impreso. Con un telefonazo o unas líneas por carta es suficiente. Especificad si queréis las dos versiones o sólo la de QL.

El sistema junto con sus fuentes ocupa un disquete de alta densidad para QL y otro para PC, de modo que si no podéis usar disquetes HD me lo tenéis que decir para que os ponga los ficheros en dos disquetes DD. Pero si pasáis de los fuentes, no hay problema, en un solo disquete DD cabe todo.

Y os advierto con toda mi buena intención: continuará...

Pedro Reina, J.13.1.1994

```
#####
#   #  #    ###  ##   ##  ###   ####
#   #  #     #   # # # #  #  #  #  #
#   #  #     #   #  #  #  ###   #  #
#####  ###  ###  #     #  #     ####
```

```
-----------------------------------------------------------------
Sistema de programación en C multiplataforma orientado al objeto
-----------------------------------------------------------------
Versión 1.1               Pedro Reina               D.9.1.1994
-----------------------------------------------------------------
```

Indice
======

Presentación

Existen en el mercado muchas herramientas de programación multiplataforma. La mayor parte de ellas son para programar en C o en C++. Las más baratas permiten compilar para MS-DOS, Windows y McIntosh; las más caras también generan ejecutables para UNIX, en sus distintas versiones, y para VMS.

Olimpo también es una herramienta multiplataforma, que se puede utilizar en QL y en PC, tanto en modo texto como en modo gráfico. Permite desarrollar código en C, y es, igual que las herramientas comerciales, orientada al objeto.

Ha sido desarrollada íntegramente en español, tanto la documentación como los fuentes y los mensajes en pantalla.

El sistema Olimpo pretende, por tanto, facilitar la tarea de los programadores en C que utilizan el español como lengua corriente. Las áreas en las que su utilización puede ser más provechosa son:

* Aumentar la legibilidad del código.
* Acortar el tiempo de desarrollo de aplicaciones.
* Reutilizar código.
* Transportar programas entre plataformas.

He decidido implementar el sistema orientado a objetos por ser el método de programación más flexible, potente y fácil de usar que conozco.

Se comienza con objetos de bajo nivel, que se comunican directamente con el hardware y hay que reescribir cuando se intenta llevar el sistema a una plataforma distinta. Estos objetos permiten uniformizar el tratamiento de las funciones de bajo nivel y sobre ellos se puede construir objetos más abstractos que dan servicios más complejos. En un futuro, pretendo crear objetos de más alto nivel, que se encarguen ellos mismos de parte de la gestión del programa.

Olimpo está disponible para el QL usando el compilador C68 y para PC usando Turbo C++ como compilador de C. Pretendo escribir una versión sobre UNIX, usando COHERENT.

El sistema consta de un fichero de cabecera, Olimpo.h, y un fichero de librería, Olimpo.lib (o libOlimpo_a, en el QL).

El sistema es de dominio público. Se puede distribuir libremente siempre y cuando se copie completo. También es posible que algún desarrollador desee entregar los fuentes de su programa, pero sin cargar el disquete con todo el sistema Olimpo. En ese caso, se puede incluir sólo el fichero de cabecera y el de librería, pero hay que hacer constar que se ha usado Olimpo, y cómo se puede obtener en su versión completa. En definitiva, mi deseo es que lo utilice quien quiera de modo que cada vez más gente conozca su existencia y lo pueda conseguir si lo desea.

Cualquier sugerencia, comentario o pregunta será bien recibido. Estos son mis datos:

Pedro Reina
c/ Sandoval 6
28010 Madrid
España

Teléfono: 91 - 593 81 59

Observación

A lo largo de la documentación se utiliza como separador entre el nombre de un fichero y su extensión el carácter '.', el habitual en UNIX y MS-DOS. En el caso del QDOS, el separador que realmente se utiliza es '_'.

Ficheros del disquete

Están disponibles dos disquetes: uno para QL y otro para PC. La relación de los ficheros esenciales es la misma en los dos casos:

```
Olimpo.txt
Olimpo.h
Olimpo.lib     (En el QL, libOlimpo_a)
Demo.c
Demo.cnf
Demo.dbf
Demo.exe
```

Olimpo.txt contiene esta documentación.
Olimpo.h es el fichero de cabecera.
Olimpo.lib es el fichero de librería.
Demo.c es el código de un programa de demostración del sistema.
Demo.cnf es el fichero de configuración de Demo.
Demo.dbf es una base de datos.
Demo.exe es el ejecutable que permite ver una pequeña demostración de las posibilidades de Olimpo.

El directorio Fuente contiene todos los ficheros fuente del sistema. Son idénticos para los dos ordenadores.

El directorio Valida contiene los ficheros que se utilizan para validar las funciones del sistema según se va desarrollando y los "makefiles" correspondientes. Los fuentes de validación son idénticos para ambos ordenadores, los makefiles no.

El directorio Uti contiene varias utilidades para crear el sistema, que dependen del ordenador.

Uso del sistema

Para utilizar el sistema hay que instalarlo primero. He intentado que esto sea lo más sencillo posible. De hecho, prácticamente no es necesaria instalación, ya que para compilar un programa con Olimpo basta incluir Olimpo.h y al invocar el montador, añadir Olimpo.lib. Para conseguir que esto se realice fácilmente, lo mejor es copiar Olimpo.h en el directorio de ficheros se cabecera del compilador (normalmente, INCLUDE) y Olimpo.lib en el de ficheros de librería (usualmente, LIB)

Los compiladores que uso con Olimpo son Turbo C++ versión 1.01 en el PC y C68 versión 4.03 en el QL. Normalmente, el uso de compiladores de versiones superiores no debería dar ningún problema.

Para compilar un programa usando Olimpo:

1. Escribir antes del código:

```
#include <Olimpo.h>
```

2. Si se utiliza el objeto Pantalla o cualquier otro que lo use, hay que empezar por usar la función Pan_Define() y terminar con Pan_Cierra(); este es el esquema general:

```
   /* Tu código */
Pan_Define (PAN_TEXTO);
   /* Tu código */
Pan_Cierra();
```

```
/* Tu código */
```

3. Al montar, añadir Olimpo.lib.

Con Turbo C, se puede usar esta orden:
TCC Programa Olimpo.lib

Con C68, ésta:
EX CC; "Programa_c -lOlimpo"

La librería suministrada en la versión PC corresponde al modelo de memoria "Small"; si se desea utilizar otro modelo, se puede volver a compilar la librería. En la sección "Cómo hacer cambios" se explica el método.

Siguiendo estas instrucciones, el programa "Hola, mundo" se escribiría así en el sistema Olimpo:

```
#include <Olimpo.h>
main()
  {
  Pan_Define (PAN_TEXTO);
  Pan_PonTexto (10,30,"Hola, mundo");
  Usr_PulsaUnaTecla ("Programa concluido");
  Pan_Cierra ();
  return (0);
  }
```

Para ejecutar los programas producidos con la ayuda de Olimpo:

En el PC:

Si se utiliza la pantalla en modo texto, no hay ninguna restricción.
Si se utiliza la pantalla en modo gráfico, el PC debe tener tarjeta VGA o compatible.

En el QL:

La pantalla debe estar en modo de alta resolución (MODE 4).
En principio Olimpo no está pensado para que sus programas operen en multitarea, pero se puede conseguir si el programador es cuidadoso.

Descripción del sistema

El sistema está distribuido por objetos. Cada uno se ocupa de una parte de la programación, y uno de los criterios de desarrollo es que sean lo más independientes entre sí que se pueda. Los objetos de alto nivel deben usar los de bajo nivel, pero nunca al revés.

Cada objeto tiene un identificador de tres letras y todas las funciones, macros y variables globales de ese objeto comienzan con ese identificador.

También hay un fichero de definiciones generales, que aunque no es un objeto propiamente dicho, conviene considerarlo como tal a efectos de documentación.

A continuación describo cada objeto:

General (Gen)

La especificación del ordenador objetivo se hace mediante la definición del macro OLIMPO_PC o del OLIMPO_QL. Se pueden utilizar para escribir distintas versiones del código que lo exija y, mediante compilación condicional, mantener un único fuente.

Se define el tipo de datos "logico", con sus valores SI y NO.

Se define el tipo de datos "contador", con un rango mínimo de valores de -32768 a 32767 (en el QL el rango es mayor).

Se define el tipo de datos "entero", con un rango de valores de -2147483648 a 2147483647.

Se define el tipo de datos "octeto", con un rango de valores de 0 a 255.

Se define la constante NULO, que es un alias de NULL (carácter nulo).

Se define la constante NIL, que es el 0 (se utiliza en LISP como fin de lista, y en ese sentido lo voy a usar en Olimpo).

Se definen las funciones Abs(), Max() y Min().

Pantalla (Pan)

Este objeto permite controlar la apariencia completa de la pantalla.

La función Pan_Define() permite iniciar los valores por defecto necesarios. En esta versión se utiliza un parámetro, que puede ser PAN_TEXTO o PAN_GRAFICO, para indicar al PC si se desea tener la pantalla en modo texto o en modo gráfico. Esto permite incluir gráficos en el PC. Este debe tener tarjeta VGA. Si no se dispone de ella, el sistema arranca en modo texto.

La función Pan_Cierra() se utiliza normalmente al final de la aplicación para restaurar todo y volver al sistema operativo.

Es perfectamente posible usar en una parte central del programa Pan_Cierra() y luego otra vez Pan_Define() para cambiar de modo, aunque el contenido de la pantalla se perderá.

En cualquier momento se puede consultar el modo en que se encuentra la pantalla mediante la función Pan_Modo().

La pantalla es de 24 filas numeradas de 0 a 23 comenzando por arriba y 80 columnas numeradas de 0 a 79 empezando por la izquierda.

Se dispone de 4 colores, definidos con los macros BLANCO, NEGRO, ROJO y VERDE. Se manejan con las funciones Pan_Papel(), Pan_Tinta() y Pan_Color(). Estas funciones conviene utilizarlas antes de escribir algo, ya que los colores de papel y de tinta se cambian muy a menudo al llamar a otros objetos, de modo que no hay ninguna garantía de que permanezcan como los determine el programador.

La función Pan_Cursor() controla la posición del cursor.

La función Pan_CursorVisible() determina si el cursor se ve en la pantalla o no. En el PC en modo gráfico no está disponible el cursor.

Las funciones Pan_Texto(), Pan_Entero() y Pan_Caracter() permiten escribir en la pantalla texto (cadenas), enteros de tipo "contador" (no de tipo "entero") y caracteres.
Las funciones Pan_PonTexto(), Pan_PonEntero() y Pan_PonCar() colocan el cursor en la posición indicada y luego escriben el texto, entero o carácter.

Se puede activar o desactivar el modo de escritura resaltada con la función Pan_Resalta(). La situación normal es la de escritura resaltada desactivada, así que es importante conectarla cuando se desee usar e, inmediatamente, desconectarla. En el PC en modo gráfico no se dispone de esta posibilidad.

La función Pan_Limpia() borra la pantalla completa del color actual del papel.

La función Pan_BorraLinea() borra una línea con el color indicado.

La función Pan_Borra() borra una zona rectangular con el color indicado.

Zona (Zon)

Este objeto es el encargado de los gráficos. Sólo se puede utilizar cuando la pantalla está en modo gráfico. Por lo tanto, en un PC que no disponga de tarjeta gráfica VGA no se puede usar este objeto.

Hay un tipo de datos llamado "zona" que permite manejar las posibilidades del objeto.

Lo primero que hay que hacer es crear una zona mediante la función Zon_Crea(). Si no se puede crear, por falta de memoria o porque la pantalla no está en modo gráfico, esta función devuelve NIL.

A partir de ese momento, podemos acceder a cada pixel de la zona.

Las funciones Zon_Alto() y Zon_Ancho() dan las dimensiones de una zona en pixels. Obviamente, la misma zona no tendrá las mismas dimensiones en ordenadores distintos, de ahí la importancia de disponer de estas funciones.

Los pixels de la zona se numeran empezando en 0 de arriba hacia abajo y de izquierda a derecha, como es lo habitual.

Se puede dibujar cada pixel de la zona mediante dos funciones, Zon_Pixel() y Zon_PixelSeguro(). La diferencia entre ambas es que la primera comprueba que el pixel pedido realmente existe en esa zona (si no existe, no lo dibuja) y la segunda no. Por tanto, la primera es más lenta; pero para usar la segunda hay que estar seguro de que el pixel pedido pertenece a la zona.

Manejar una pantalla pixel a pixel es bastante lento, de modo que he buscado una manera más rápida de manejar físicamente las zonas. El resultado de la búsqueda ha sido el manejo de cada "oxel" de la zona. Tanto la tarjeta VGA como la memoria gráfica del QL manejan los pixels agrupados de ocho en ocho. Pues bien, cada grupo de 8 pixels manejado globalmente por el hardware es lo que he denominado oxel. Siempre agrupa 8 pixels en línea, nunca en columnas.

Se puede saber cuántos oxels de ancho tiene una zona se usa la función Zon_AnchoEnOxel(). La altura de una zona no se puede medir en oxels.

Para manejar cada oxel de una zona se usan las funciones Zon_Oxel() y Zon_OxelSeguro(). La diferencia entre ambas es la misma que para los pixels. Estas dos funciones necesitan conocer no sólo el color que hay que usar, sino también cuáles de los pixels del oxel se desean poner de ese color. Esto se indica mediante las "máscaras". Una máscara es simplemente un octeto, ocho bits. Cuando el bit es un 1, el pixel correspondiente cambia al color pedido; cuando es 0, no cambia. Por ejemplo, si deseamos cambiar de color el pixel más a la izquierda y el más a la derecha de un oxel, la máscara debe ser 10000001, que en decimal es 129, y en hexadecimal 0x81.

Cuando una zona ya no sea necesaria, hay que liberar la memoria que usa con la función Zon_Destruye().

Azar (Azr)

Se pueden obtener números enteros aleatorios en el margen que se desee. Basta iniciar el generador al principio con Azr_Inicia() y luego ir pidiendo enteros con Azr_Entero().

Tiempo (Tim)

La función Tim_Crono() devuelve una referencia temporal de precisión de tipo "tiempo". Si se restan dos valores devueltos por Tim_Crono(), se obtiene el tiempo real transcurrido entre las dos llamadas.

La función Tim_Espera() obliga al programa a suspender su ejecución un periodo de tiempo determinado.

Se puede consultar el reloj del sistema, viendo el año, mes y día en curso.

Cadena (Cad)

El manejo de cadenas en C no es muy cómodo. En esta implementación me he decidido por utilizar cadenas de asignación dinámica, por ser una aproximación más flexible que las cadenas de asignación estática.

He definido el tipo "cadena", que es simplemente un puntero a char; es decir, si declaramos una variable como tipo cadena, no se reserva automáticamente memoria para ella, sino que hay que pedirla explícitamente y luego liberarla. Muchas de las funciones definidas en este objeto y otros que lo utilizan realizan la reserva de memoria sin que el programador se tenga que ocupar.

Esto permite trabajar de dos formas distintas con las cadenas:

Por ser punteros, se pueden asignar a cadenas constantes. Por ejemplo, es válido:

```
cadena Cad;
Cad = "Esto es una cadena";
  /* Aquí tu código */
Cad = "Y esto es otra cadena";
```

Y por otro lado, se pueden crear cadenas de la longitud precisa en cada momento:

```
cadena Cad;
Cad = Cad_Crea (10);
Cad_Copia (Cad,"0123456789");
  /* Aquí tu código */
Cad_Destruye (Cad);
```

Las funciones más importantes son las que reservan memoria y la liberan.

Cad_Crea() reserva memoria para una cadena de la longitud indicada (reservando un octeto más para el carácter NULL), le da el valor inicial de cadena vacía y devuelve la dirección de memoria reservada (que es un puntero a char). La memoria así reservada no se libera automáticamente al salir de la función en la que se reserva, de modo que es responsabilidad del programador destruirla cuando no sea necesaria.

Cad_Destruye() es la encargada de liberar la memoria reservada mediante Cad_Crea(). También se debe usar para liberar memoria reservada por otras funciones.

Lista (Lis)

Las listas encadenadas son una de las estructuras más importantes en informática. Olimpo proporciona una implementación muy sencilla de las listas.

Esta es la estructura general de una lista:

```
  -------------           -------------                  -------------
  | Contenido |           | Contenido |                  | Contenido |
  |-----------|    /-->   |-----------|    /--> ... -->  |-----------|
  | Siguiente | --/       | Siguiente | --/              | Siguiente | --> NIL
  -------------           -------------                  -------------
```

El tipo "lista" corresponde a estructuras como la de arriba. Con Lis_Crea() se obtiene una lista con un solo nodo que no tiene contenido y que tiene como siguiente nodo NIL, es decir: ninguno (o "tierra", como se dice a veces).

El "contenido" puede ser cualquier cosa, ya que lo que realmente se almacena es

un puntero a una dirección de memoria que haya sido reservada mediante malloc(); por ejemplo, se puede almacenar una cadena creada por Cad_Crea(). Para anotar en el nodo recién creado el contenido que se desee está la función Lis_PonContenido().

A continuación hay que ir añadiendo más nodos a la lista. Esto es más sencillo, porque las funciones Lis_Agrega() y Lis_AgregaFin() sólo necesitan recibir la lista a la que hay que añadir el nodo y el contenido del nuevo nodo; ellas se encargar de crear el nuevo nodo, poner el contenido, añadir el nuevo nodo y devuelven la lista resultante.

Para usar la lista hay que ir viendo su contenido con Lis_Contenido() y navegar por la lista usando Lis_Siguiente().

Cuando la lista no sea necesaria, se libera su memoria con Lis_Destruye(), a la que basta darle el primer nodo, ya que ella va recorriendo toda la lista liberando la memoria reservada para todos los nodos.

Sonido (Son)

Este objeto se encarga de las señales acústicas. Se pueden conectar con Son_Enciende(), desconectar con Son_Apaga() o cambiar de estado con Son_Cambia().

Las llamadas a las funciones de este objeto no provocan inmediatamente la generación del sonido, eso dependerá de si en ese momento está conectado o no.

Tecla (Tec)

He definido el tipo "tecla" y la función Tec_Pulsada() como básicos para poder olvidar las grandes diferencias en el tratamiento de las teclas en los diversos sistemas operativos. En Olimpo sólo hay que saber que Tec_Pulsada() devuelve la tecla que haya pulsado el usuario, que pertenece al tipo "tecla". Junto con eso, hay toda una colección de macros que definen cada tecla; por ejemplo, la tecla F1 está definida con el macro TEC_F1, y el programador no necesita conocer qué codigo (o códigos) asigna el sistema operativo a F1 (cada uno lo hace de una forma distinta). En esta versión no dispongo de macros para la ñ ni las vocales acentuadas.

Se puede consultar en cualquier momento cuál fue la última tecla pulsada por el usuario con Tec_Ultima(), lo que resulta muy cómodo para obtener información en funciones muy alejadas de aquellas en las que el programa interactúa con el usuario.

Se puede pedir una tecla que pertenezca a un determinado rango con Tec_Validada() e incluso hacerlo con vuelta inmediata, si no hay tecla correcta disponible, con Tec_ValidadaRapido().

Hay funciones para pasar a mayúsculas o minúsculas, pero en esta versión sólo para caracteres ASCII (hasta 127).

No he definido macros para todas las teclas que son posibles en cada ordenador, sino sólo para aquellas que son perfectamente compatibles. En el PC existen algunas teclas que no están disponibles en el QL, pero con funciones que normalmente en el QL se asignan a determinadas combinaciones de tecla.

Esta es la lista de los macros que corresponden a distintas pulsaciones en PC y en QL:

Macro	Tecla PC	Tecla QL
TEC_INICIO	Inicio	ALT/Cursor izquierda
TEC_FIN	Fin	ALT/Cursor derecha
TEC_AVPAG	AvPág	ALT/Cursor abajo
TEC_REPAG	RePág	ALT/Cursor arriba
TEC_CTRL_INICIO	CTRL/Inicio	CTRL/ALT/Cursor izquierda

```
TEC_CTRL_FIN          CTRL/Fin          CTRL/ALT/Cursor derecha
TEC_CTRL_AVPAG        CTRL/AvPág        CTRL/ALT/Cursor abajo
TEC_CTRL_REPAG        CTRL/RePág        CTRL/ALT/Cursor arriba
TEC_INSERTAR          Insert            MAY/CAPSLOCK
TEC_SUPRIMIR          Supr              CTRL/Cursor derecha
TEC_RETROCESO         Retroceso         CTRL/Cursor izquierda
```

Y esta es la de teclas comunes:

```
TEC_ESC  TEC_ENTER  TEC_ESPACIO
TEC_ARRIBA  TEC_ABAJO  TEC_IZQUIERDA  TEC_DERECHA
TEC_TAB  TEC_MAY_TAB
TEC_A ... TEC_Z
TEC_MAY_A ... TEC_MAY_Z
TEC_0  TEC_1 ... TEC_9
TEC_CTRL_A ... TEC_CTRL_Z
TEC_ALT_A ... TEC_ALT_Z
TEC_ALT_0  TEC_ALT_1 ... TEC_ALT_9
TEC_F1  TEC_F2  TEC_F3  TEC_F4  TEC_F5
TEC_MAY_F1  TEC_MAY_F2  TEC_MAY_F3  TEC_MAY_F4  TEC_MAY_F5
TEC_CTRL_F1  TEC_CTRL_F2  TEC_CTRL_F3  TEC_CTRL_F4
TEC_ALT_F1  TEC_ALT_F2  TEC_ALT_F3  TEC_ALT_F4  TEC_ALT_F5
```

Cuadro (Cdr)

Se pueden representar cuadros, cajas y líneas de cualquier color disponible y en dos grosores distintos.

El color se indica con Cdr_Color() y para representar los grosores de las líneas se tienen los macros CDR_SIMPLE y CDR_DOBLE.

Usuario (Usr)

Las dos últimas filas de la pantalla están reservadas para interactuar con el usuario. Hay funciones para darle indicaciones, informarle de acciones y también para que edite enteros y texto (cadenas).

Fichero (Fch)

La función Fch_AbreLeer() permite abrir un fichero para leer datos de él, la función Fch_AbreGrabar() abrirlo para grabar datos y la Fch_AbreActualizar() abrirlo para leer sus datos y modificarlos. Ambas cosas se pueden hacer en modo texto y en modo binario, y para indicarlo están los macros FCH_TEXTO y FCH_BINARIO. Naturalmente, esta distinción es necesaria por culpa del MS-DOS; de hecho, en el QL no hay ninguna diferencia entre los dos modos.

Una vez abierto un fichero, se pueden leer y escribir líneas con Fch_LeeLinea() y Fch_EscribeLinea() y grupos de octetos con Fch_LeeOcteto() y Fch_EscribeOcteto().

Se puede mover la posición de lectura y escritura en el fichero con las funciones Fch_Coloca() y Fch_ColocaFinal().

Las funciones se encargan de gestionar los errores que se puedan producir. Por ejemplo, al llamar a Fch_Borra(), primero se pide confirmación al usuario, luego se comprueba la existencia del fichero y por último se emite un mensaje indicando si ha sido posible el borrado o no.

Menú (Men)

Hay menús horizontales y verticales. Sólo hay que preocuparse de asignar la zona de la pantalla en la que se desea que aparezca el menú, el objeto se ocupa de la

colocación. Las opciones pueden llevar una tecla caliente, un atajo o "hotkey", que permite elegir la opción simplemente pulsando esa tecla. Para indicar cuál es, se utiliza el carácter '>' antes de la letra en la llamada al objeto.

Sólo hay dos funciones para manejar menús: Men_Horizontal() y Men_Vertical().

Si a un menú se le mandan más opciones de las que puede manejar, el exceso de opciones será simplemente ignorado.

Programa (Prg)

Este objeto se ocupa simplemente de presentar información sobre el programa en la línea superior de la pantalla. Si no se utiliza, se dispone de una línea más para el desarrollo del programa.

La función que presenta esta información es Prg_Presenta().

Configuración (Cnf)

Este objeto permite leer información de un fichero filtrando todos los comentarios que se hayan escrito en él. El programador debe decidir después cómo lee la información entregada por el objeto, que lo hace en forma de cadena.

En los ficheros de configuración se considera línea de comentario toda aquella que tenga un '*' (asterisco) como primer carácter, aunque haya caracteres en blanco antes de él.

También se pueden poner comentarios dentro de una línea poniendo "//" (doble barra). Todo lo que haya a continuación, se considera comentario.

Las líneas en blanco se ignoran.

Un fichero se abre en modo configuración con la función Cnf_Abre(), se va leyendo línea a línea con Cnf_Lee() y por fin se cierra con Cnf_Cierra().

Base de datos (Bdt)

Una base de datos se puede crear con la función Bdt_Crea() y abrir con Bdt_Crea(). Estas funciones devuelven un tipo "basedato", que es necesario usar en las futuras referencias a la base de datos.

Una vez creada o abierta es posible acceder a sus registros. Cada registro se mantiene en memoria, de modo que se pueden leer y cambiar sus campos.

También se puede examinar la estructura de la base de datos, viendo las características de cada campo.

Cuando se termine de usar una base de datos, se debe cerrar con Bdt_Cierra(). Esto es imprescindible, ya que si no se hace algunos datos no quedarán actualizados en el fichero correspondiente.

¿Y ahora qué mas?

Para ampliar información sobre el uso de los objetos, consulta la relación alfabética de funciones y mira el fichero de demostración (Demo.c) y los de validación de funciones (V*.c)

Relación alfabética de funciones

```
FUNCION:  Abs()
OBJETIVO: Calcular el valor absoluto de un número
ENTRADAS: Un número
SALIDAS:  El valor absoluto
EJEMPLO:  Abs(-3)

FUNCION:  Azr_Entero()
OBJETIVO: Obtener un número entero aleatorio en un rango especificado
ENTRADAS: a: número más bajo admitido; b: el más alto admitido
SALIDAS:  un número entero entre a y b (ambos inclusive)
EJEMPLOS: Azr_Entero (0,100)
          Azr_Entero (-10,10)

FUNCION:  Azr_Inicia()
OBJETIVO: Iniciar el generador con un número aleatorio
ENTRADAS: Ninguna, se lee el reloj del sistema
SALIDAS:  Ninguna
EJEMPLO:  Azr_Inicia ()

FUNCION:  Bdt_ActivoRegistro()
OBJETIVO: Decir si el registro en memoria está activo (es decir:
          no está marcado como borrado)
ENTRADAS: La base de datos
SALIDAS:  Lógica indicando si el registro está activo
EJEMPLO:  Bdt_ActivoRegistro (Agenda)

FUNCION:  Bdt_Actual()
OBJETIVO: Decir el número de registro del registro actual de una
          base de datos
ENTRADAS: La base de datos
SALIDAS:  El entero que indica el número del registro actual
EJEMPLO:  Bdt_Actual ( Agenda )

FUNCION:  Bdt_AgregaRegistro()
OBJETIVO: Agregar un registro desde memoria al fichero, poniéndolo
          al final de éste
ENTRADAS: La base de datos y una variable donde dejar el
          número de registro
SALIDAS:  Lógica indicando si el registro se ha escrito. En la variable
          queda indicado el número de registro
NOTA:     Los registros se cuentan a partir de 0
EJEMPLO:  Bdt_AgregaRegistro (Agenda,&Numero)

FUNCION:  Bdt_AnoUltimo()
OBJETIVO: Decir el año de la última modificación de una base de datos
ENTRADAS: La base de datos
SALIDAS:  El contador que indica el año
EJEMPLO:  Bdt_AnoUltimo ( Agenda )

FUNCION:  Bdt_BorraRegistro()
OBJETIVO: Marcar un registro como borrado
ENTRADAS: La base de datos y el número de registro
SALIDAS:  Lógica indicando si el registro se ha borrado
NOTAS:    1. Los registros se cuentan a partir de 0
          2. El registro no se borra físicamente, sólo se marca
          3. El registro actual en memoria queda éste
EJEMPLO:  Bdt_BorraRegistro (Agenda,1)

FUNCION:  Bdt_CampoNombre()
OBJETIVO: Dar el contenido de un campo a partir de su nombre
ENTRADAS: La base de datos, el nombre del campo y una cadena donde
          dejar el contenido
SALIDAS:  Lógica indicando si el campo existe. La cadena queda rellena
NOTA:     La función no comprueba la longitud de la cadena; el
          programador debe asegurarse de que es suficientemente
          larga para contener el contenido del campo
EJEMPLO:  Bdt_CampoNombre (Agenda,"EDAD",Aux)

FUNCION:  Bdt_CampoNumero()
```

```
      OBJETIVO: Dar el contenido de un campo a partir de su número
      ENTRADAS: La base de datos, el número del campo y una cadena donde
                dejar el contenido
      SALIDAS:  Lógica indicando si el campo existe. La cadena queda rellena
      NOTAS:    1. Los campos se numeran a partir de 0
                2. La función no comprueba la longitud de la cadena; el
                   programador debe asegurarse de que es suficientemente
                   larga para contener el contenido del campo
      EJEMPLO:  Bdt_CampoNumero (Agenda,1,Aux)

      FUNCION:  Bdt_Crea()
      OBJETIVO: Crear un fichero de base de datos y obtener un objeto
                en memoria para manejarlo
      ENTRADAS: El nombre completo del fichero que hay que crear, un vector
                con los nombres de los campos (terminado en NIL), un vector
                de caracteres con los tipos de los campos, un vector de números
                con las longitudes de cada campo y un vector con la cantidad de
                decimales de cada campo.
      SALIDAS:  Una base de datos o NIL si no se puede crear
      EJEMPLO:  Bdt_Crea ("Agenda.dbf", {"NOMBRE", "EDAD", NIL}, {'C','N'},
                          {20, 3}, {0, 0})
      NOTAS:    1. Los nombres de los campos no pueden tener más de 10 caracteres
                2. Es costumbre que los nombres de los campos se escriban en
                   mayúsculas
                3. Los tipos de campo pueden ser:
                   'C' -> carácter
                   'D' -> fecha, debe tener longitud 8
                   'L' -> lógico, debe tener longitud 1
                   'M' -> memo, debe tener longitud 10
                   'N' -> numérico

      FUNCION:  Bdt_DecimalDeCampo()
      OBJETIVO: Decir el número de decimales de un campo a partir de su número
      ENTRADAS: La base de datos y el número de campo
      SALIDAS:  Un número indicando el número de decimales del campo.
                Si el campo no existe, se devuelve 0
      NOTA:     Los campos se numeran desde 0
      EJEMPLO:  Bdt_DecimalDeCampo (Agenda,1)

      FUNCION:  Bdt_DiaUltimo()
      OBJETIVO: Decir el día de la última modificación de una base de datos
      ENTRADAS: La base de datos
      SALIDAS:  El contador que indica el día
      EJEMPLO:  Bdt_DiaUltimo ( Agenda )

      FUNCION:  Bdt_LeeRegistro()
      OBJETIVO: Leer un registro desde el fichero a memoria
      ENTRADAS: La base de datos y el número de registro
      SALIDAS:  Lógica indicando si el registro se ha leído. La zona definida
                por InforActual queda cambiada
      NOTA:     Los registros se cuentan a partir de 0
      EJEMPLO:  Bdt_LeeRegistro (Agenda,1)

      FUNCION:  Bdt_LimpiaRegistro()
      OBJETIVO: Limpiar el registro en memoria
      ENTRADAS: La base de datos
      SALIDAS:  Ninguna
      EJEMPLO:  Bdt_LimpiaRegistro (Agenda)

      FUNCION:  Bdt_LongitudDeCampo()
      OBJETIVO: Decir la longitud de un campo a partir de su número
      ENTRADAS: La base de datos y el número de campo
      SALIDAS:  Un número indicando la longitud del campo. Si el campo no
                existe, se devuelve 0
      NOTA:     Los campos se numeran desde 0
      EJEMPLO:  Bdt_LongitudDeCampo (Agenda,1)

      FUNCION:  Bdt_MesUltimo()
      OBJETIVO: Decir el mes de la última modificación de una base de datos
      ENTRADAS: La base de datos
```

```
      SALIDAS:  El contador que indica el mes
      EJEMPLO:  Bdt_MesUltimo ( Agenda )

      FUNCION:  Bdt_NombreDeCampo()
      OBJETIVO: Decir el nombre de un campo a partir de su número
      ENTRADAS: La base de datos, el número de campo y la cadena donde
                dejar el nombre
      SALIDAS:  Lógica indicando si el campo existe. La cadena queda rellena
      EJEMPLO:  Bdt_NombreDeCampo (Agenda,1,Aux)
      NOTAS:    1. La cadena reservada debe tener 11 octetos (o más)
                2. Los campos se numeran desde 0

      FUNCION:  Bdt_NumeroDeCampo()
      OBJETIVO: Hallar el número de un campo a partir de su nombre
      ENTRADAS: La base de datos, una variable donde dejar el número y
                una cadena indicando el nombre
      SALIDAS:  Lógica indicando si el campo existe. El número queda relleno
      EJEMPLO:  Bdt_NumeroDeCampo (Agenda,&Numero,Nombre)

      FUNCION:  Bdt_PonCampoNombre()
      OBJETIVO: Poner el contenido de un campo dado su nombre
      ENTRADAS: La base de datos, el nombre del campo y una cadena con
                el contenido
      SALIDAS:  Lógica indicando si el campo existe.
      NOTA:     La función no comprueba la longitud de la cadena; el
                programador debe asegurarse de que no es
                más larga que la longitud del campo
      EJEMPLO:  Bdt_PonCampoNombre (Agenda,"EDAD"," 35")

      FUNCION:  Bdt_PonCampoNumero()
      OBJETIVO: Poner el contenido de un campo dado su número
      ENTRADAS: La base de datos, el número del campo y una cadena con
                el contenido
      SALIDAS:  Lógica indicando si el campo existe.
      NOTAS:    1. Los campos se numeran a partir de 0
                2. La función no comprueba la longitud de la cadena; el
                   programador debe asegurarse de que no es
                   más larga que la longitud del campo
      EJEMPLO:  Bdt_PonCampoNumero (Agenda,1,"Alberto")

      FUNCION:  Bdt_RecuperaRegistro()
      OBJETIVO: Recuperar un registro que estaba marcado como borrado
      ENTRADAS: La base de datos y el número de registro
      SALIDAS:  Lógica indicando si el registro se ha recuperado
      NOTAS:    1. Los registros se cuentan a partir de 0
                2. El registro actual en memoria queda éste
      EJEMPLO:  Bdt_RecuperaRegistro (Agenda,1)

      FUNCION:  Bdt_TipoDeCampo()
      OBJETIVO: Decir el tipo de un campo a partir de su número
      ENTRADAS: La base de datos y el número de campo
      SALIDAS:  Un carácter indicando el tipo de campo. Si el campo no
                existe, se devuelve NULO
      NOTA:     Los campos se numeran desde 0
      EJEMPLO:  Bdt_TipoDeCampo (Agenda,1)

      FUNCION:  Bdt_TotalCampo()
      OBJETIVO: Decir el número de campos de una base de datos
      ENTRADAS: La base de datos
      SALIDAS:  El contador que indica el número de campos
      EJEMPLO:  Bdt_TotalCampo ( Agenda )

      FUNCION:  Bdt_TotalRegistro()
      OBJETIVO: Decir el número de registros de una base de datos
      ENTRADAS: La base de datos
      SALIDAS:  El entero que indica el número de registros
      EJEMPLO:  Bdt_TotalRegistro ( Agenda )

      FUNCION:  Cad_Cambia()
      OBJETIVO: Cambiar en una cadena un carácter por otro
```

```
      ENTRADAS: Cadena destino, carácter viejo, carácter nuevo
      SALIDAS:  La cadena queda modificada y se devuelve
      EJEMPLO:  Cad_Cambia (Cadena,'a','b')

      FUNCION:  Cad_CarPertenece()
      OBJETIVO: Averiguar si un caracter pertenece a una cadena
      ENTRADAS: La cadena y el caracter
      SALIDA:   La posición que ocupa el carácter en la cadena, o 0
      NOTA:     Se empieza a contar en 1
      EJEMPLO:  Cad_CarPertenece (Nombre,'a')

      FUNCION:  Cad_Copia()
      OBJETIVO: Copiar una cadena en otra
      ENTRADAS: La cadena destino y la cadena origen
      SALIDAS:  La cadena destino
      EJEMPLO:  Cad_Copia (Nombre,"Eustaquio")

      FUNCION:  Cad_Crea()
      OBJETIVO: Reservar memoria para almacenar una cadena
      ENTRADAS: El número máximo de caracteres de la cadena
      SALIDAS:  Una cadena
      NOTAS:    Se reserva espacio automáticamente para el NULO
                La cadena queda iniciada a cadena vacía
                La cadena devuelta hay que destruirla cuando no sea necesaria
      EJEMPLO:  Cad_Crea (80)

      FUNCION:  Cad_Destruye()
      OBJETIVO: Liberar la memoria reservada para una cadena
      ENTRADAS: La cadena
      SALIDAS:  Ninguna
      EJEMPLO:  Cad_Destruye (Cadena)

      FUNCION:  Cad_Entero()
      OBJETIVO: Convertir un número entero en una cadena
      ENTRADAS: El número
      SALIDAS:  La cadena
      NOTA:     La cadena devuelta hay que destruirla cuando no sea necesaria
      EJEMPLO:  Cad_Entero (130)

      FUNCION:  Cad_Igual()
      OBJETIVO: Indicar si dos cadenas son iguales
      ENTRADAS: Las dos cadenas
      SALIDA:   Lógica
      EJEMPLO:  Cad_Igual (Nombre,"Eustaquio")


      FUNCION:  Cad_Longitud()
      OBJETIVO: Calcular la longitud de una cadena
      ENTRADAS: Una cadena
      SALIDAS:  Su longitud
      EJEMPLO:  Cad_Longitud (Nombre)

      FUNCION:  Cad_Mueve()
      OBJETIVO: Mover hacia la derecha todos los caracteres de una cadena
                desde una posición determinada, cuidando que la longitud
                no exceda un cierto máximo
      ENTRADAS: La cadena, la posición y el máximo
      NOTA:     Las posiciones se cuentan a partir de cero
      SALIDAS:  La cadena queda modificada y se devuelve
      EJEMPLO:  Cad_Mueve (Cadena,2,40)

      FUNCION:  Cad_PrimerUtil()
      OBJETIVO: Decir la posición del primer carácter de una cadena que no
                sea ni blanco, ni tabulador, ni retorno de carro ni nueva línea
      ENTRADAS: La cadena
      SALIDA:   La posición que ocupa el primer carácter útil, o 0
      NOTA:     Se empieza a contar en 1
      EJEMPLO:  Cad_PrimerUtil ( "   Ahora" )

      FUNCION:  Cad_QuitaCar()
```

```
OBJETIVO: Quitar el caracter que ocupa una determinada posición
          en una cadena
ENTRADAS: La cadena y la posición que hay que suprimir
NOTA:     Las posiciones se cuentan a partir de cero
SALIDAS:  La cadena queda modificada y se devuelve
EJEMPLO:  Cad_QuitaCaracter (Cadena,2)

FUNCION:  Cad_Subcadena()
OBJETIVO: Averiguar si la segunda cadena está contenida en la primera
ENTRADAS: Las dos cadenas
SALIDA:   La posición que ocupa el primer carácter de la primera donde
          comienza la primera aparición de la segunda, o 0
NOTA:     Se empieza a contar en 1
EJEMPLO:  Cad_Subcadena ( "Calamares","mar" )

FUNCION:  Cad_Trozo()
OBJETIVO: Extraer una cadena de otra
ENTRADAS: Cadena origen y puntos de comienzo y fin del corte
SALIDAS:  La cadena destino
NOTAS:    El primer carácter lleva el número 1
          La cadena devuelta hay que destruirla cuando no sea necesaria
EJEMPLO:  Cad_Trozo (Nombre,3,7)

FUNCION:  Cad_Une()
OBJETIVO: Unir varias cadenas
ENTRADAS: Las cadenas que hay que unir, terminadas por NIL
SALIDAS:  Una cadena conteniendo la unión
NOTA:     La cadena devuelta hay que destruirla cuando no sea necesaria
EJEMPLO:  Cad_Une ("Veinte de ", Mes, " de 1808",NIL)

FUNCION:  Cdr_Caja()
OBJETIVO: Dibujar una caja
ENTRADAS: Tipo de la caja y sus coordenadas
SALIDAS:  Ninguna
EJEMPLO:  Cdr_Caja (CDR_DOBLE,1,1,10,70)

FUNCION:  Cdr_Color()
OBJETIVO: Establecer el color con el que se dibujarán todas las figuras
ENTRADAS: El color
SALIDAS:  Ninguna
EJEMPLO:  Cdr_Color ( ROJO )

FUNCION:  Cdr_Dibuja()
OBJETIVO: Dibujar un cuadro
ENTRADAS: El tipo de línea que se dibuja en las líneas externas,
          el de las internas, el número de filas, el alto de cada
          una, el número de columnas, el ancho de cada una, la fila
          y columna de la esquina superior izquierda
SALIDAS:  Ninguna
EJEMPLO:  Cdr_Dibuja (CDR_DOBLE,CDR_SIMPLE,2,3,4,1,3,3)

FUNCION:  Cdr_Linea()
OBJETIVO: Dibujar una línea
ENTRADAS: El tipo de línea y las coordenadas
SALIDAS:  Ninguna
EJEMPLO:  Cdr_Linea (CDR_DOBLE,2,3,2,71)

FUNCION:  Cnf_Abre()
OBJETIVO: Abrir un fichero de configuración
ENTRADAS: El nombre del fichero
SALIDAS:  El fichero abierto o NULO si ha habido algún error
EJEMPLO:  Cnf_Abre ( "Datos.dat" )

FUNCION:  Cnf_Cierra()
OBJETIVO: Cerrar un fichero de configuración
ENTRADAS: El fichero
SALIDAS:  NULO si todo va bien, EOF si hay error
EJEMPLO:  Cnf_Cierra ( FicheroConfig )

FUNCION:  Cnf_Lee()
```

```
      OBJETIVO: Obtener la siguiente línea de un fichero de configuración
      ENTRADAS: El fichero
      SALIDAS:  La cadena si no hay error, o NIL
      NOTAS:    La cadena no contiene '\n'
                La cadena devuelta hay que destruirla cuando no sea necesaria
      EJEMPLO:  Cnf_Lee ( Entrada )

      FUNCION:  Fch_Abre()
      OBJETIVO: Abrir un fichero
      ENTRADAS: El nombre del fichero y el tipo
      SALIDAS:  El fichero abierto o NULO si ha habido algún error
      EJEMPLO:  Fch_Abre ( "Datos.dat", "r" )

      FUNCION:  Fch_AbreActualizar()
      OBJETIVO: Abrir un fichero para actualizarlo
      ENTRADAS: El nombre del fichero y el modo: texto o binario
      SALIDAS:  El fichero abierto o NULO si ha habido algún error
      NOTA:     El puntero del fichero queda en el comienzo
      EJEMPLO:  Fch_AbreActualizar ( "Datos.dat",FCH_BINARIO )

      FUNCION:  Fch_AbreGrabar()
      OBJETIVO: Abrir un fichero para grabar en él
      ENTRADAS: El nombre del fichero y el modo: texto o binario
      SALIDAS:  El fichero abierto o NULO si ha habido algún error o negativa
      EJEMPLO:  Fch_AbreGrabar ( "Datos.bin",FCH_BINARIO )

      FUNCION:  Fch_AbreLeer()
      OBJETIVO: Abrir un fichero para leerlo
      ENTRADAS: El nombre del fichero y el modo: texto o binario
      SALIDAS:  El fichero abierto o NULO si ha habido algún error
      EJEMPLO:  Fch_AbreLeer ( "Datos.dat",FCH_TEXTO )

      FUNCION:  Fch_Borra()
      OBJETIVO: Borrar un fichero
      ENTRADAS: El nombre del fichero
      SALIDAS:  Ninguna
      EJEMPLO:  Fch_Borra ( "Viejo.txt" )

      FUNCION:  Fch_Cierra()
      OBJETIVO: Cerrar un fichero
      ENTRADAS: El fichero
      SALIDAS:  NULO si todo va bien, EOF si hay error
      EJEMPLO:  Fch_Cierra ( Salida )

      FUNCION:  Fch_Coloca()
      OBJETIVO: Colocar el puntero interno de un fichero en una determinada
                posición
      ENTRADAS: El fichero y la nueva posición, contada desde el principio
      SALIDAS:  0 si todo va bien, no cero si hay error
      EJEMPLO:  Fch_Coloca ( Dato, 50 )

      FUNCION:  Fch_ColocaFinal()
      OBJETIVO: Colocar el puntero interno de un fichero al final
      ENTRADAS: El fichero
      SALIDAS:  0 si todo va bien, no cero si hay error
      EJEMPLO:  Fch_ColocaFinal ( Dato )

      FUNCION:  Fch_EscribeLinea()
      OBJETIVO: Escribir una línea en un fichero (es decir, se pone '\n')
      ENTRADAS: El fichero y una cadena
      SALIDAS:  El fichero si no hay error, o NIL
      EJEMPLO:  Fch_EscribeLinea ( Salida, Linea )

      FUNCION:  Fch_EscribeOcteto()
      OBJETIVO: Escribir en un fichero cierta cantidad de octetos
      ENTRADAS: El fichero, la zona de memoria de donde se toman los octetos
                y la cantidad de octetos
      SALIDAS:  El fichero o NIL si ha habido algún error
      EJEMPLO:  Fch_EscribeOcteto ( Salida, Item, 1024 )
```

```
FUNCION:  Fch_Existe()
OBJETIVO: Informar si un fichero existe
ENTRADAS: El nombre del fichero
SALIDAS:  Lógica
EJEMPLO:  Fch_Existe ( "Datos.dat" )

FUNCION:  Fch_LeeLinea()
OBJETIVO: Obtener la siguiente línea de un fichero
ENTRADAS: El fichero
SALIDAS:  La cadena si no hay error, o NIL
NOTAS:    La cadena no contiene '\n'
          La cadena devuelta hay que destruirla cuando no sea necesaria
EJEMPLO:  Fch_LeeLinea ( Entrada )

FUNCION:  Fch_LeeOcteto()
OBJETIVO: Leer de un fichero cierta cantidad de octetos
ENTRADAS: El fichero, la zona de memoria donde dejar lo leido y la
          cantidad de octetos
SALIDAS:  El fichero o NIL si ha habido algún error
EJEMPLO:  Fch_LeeOcteto ( Datos, Info, 1024 )

FUNCION:  Fch_Nombre()
OBJETIVO: Formar el nombre completo de un fichero uniendo el nombre,
          el separador y la extensión
ENTRADAS: El nombre y la extensión
SALIDAS:  La cadena con el nombre completo
NOTA:     La cadena devuelta hay que destruirla cuando no sea necesaria
EJEMPLO:  Fch_Nombre ( "Datos", "txt" )

FUNCION:  Lis_Agrega()
OBJETIVO: Añadir un elemento a una lista
ENTRADAS: La lista y el contenido del nuevo elemento
SALIDAS:  Una lista nueva o la antigua si no ha sido posible realizar
          el agregado
EJEMPLO:  Lis_Agrega (Agenda,Nombre)
NOTA:     El elemento agregado se coloca el primero de la lista

FUNCION:  Lis_AgregaFin()
OBJETIVO: Añadir un elemento a una lista por el final
ENTRADAS: La lista y el contenido del nuevo elemento
SALIDAS:  La lista original
EJEMPLO:  Lis_AgregaFin (Agenda,Nombre)
NOTAS:    El elemento agregado se coloca el último de la lista
          Si no hay memoria suficiente, no se agrega

FUNCION:  Lis_Contenido()
OBJETIVO: Dar el contenido de un elemento de una lista
ENTRADAS: Una lista
SALIDAS:  El contenido del primer elemento
EJEMPLO:  Lis_Contenido (Agenda)

FUNCION:  Lis_Crea()
OBJETIVO: Crear una lista y darle valores iniciales seguros
ENTRADAS: Ninguna
SALIDAS:  Una lista nueva o NIL si no hay memoria para crearla
EJEMPLO:  Lis_Crea()

FUNCION:  Lis_Destruye()
OBJETIVO: Destruir una lista completa, incluyendo los contenidos
ENTRADAS: Una lista
SALIDAS:  Ninguna
EJEMPLO:  Lis_Destruye (Agenda)

FUNCION:  Lis_PonContenido()
OBJETIVO: Poner el contenido de un elemento de una lista
ENTRADAS: Una lista y el contenido que se va a poner, que debe ser una
          dirección de memoria reservada
SALIDAS:  El contenido
EJEMPLO:  Lis_PonContenido (Agenda,Nombre)
```

```
FUNCION:  Lis_PonSiguiente()
OBJETIVO: Poner el siguiente elemento de una lista
ENTRADAS: Una lista y la lista que se va a poner como siguiente
SALIDAS:  La lista siguiente
EJEMPLO:  Lis_PonSiguiente (Agenda,Nuevo)

FUNCION:  Lis_Siguiente()
OBJETIVO: Dar el siguiente elemento de una lista
ENTRADAS: Una lista
SALIDAS:  Otra lista, que comienza en el siguiente elemento de la
          lista original
EJEMPLO:  Lis_Siguiente (Agenda)

FUNCION:  Lis_Total()
OBJETIVO: Calcular el número de elementos de una lista
ENTRADAS: Una lista
SALIDAS:  El número de elementos
EJEMPLO:  Lis_Total (Agenda)

FUNCION:  Max()
OBJETIVO: Calcular el máximo de dos números
ENTRADAS: Los números
SALIDAS:  El mayor de ellos
EJEMPLO:  Max (2,5)

FUNCION:  Men_Horizontal()
OBJETIVO: Presentar al usuario un menú horizontal para que pueda
          elegir entre varias opciones, mediante las teclas del
          cursor o mediante un atajo (tecla caliente)
ENTRADAS: La fila en la que se muestra el menú, las columnas entre
          las que se muestra, un vector de cadenas, terminado en
          NIL, en el que se señalan los atajos y la opción que hay
          resaltar en primer lugar
SALIDAS:  Un número indicando la opción elegida, empezando a contar
          en 1, o 0 si no se elige ninguna
EJEMPLO:  Men_Horizontal (1,1,78,{">Fichero","A>yuda",NIL},2)

FUNCION:  Men_Vertical()
OBJETIVO: Presentar al usuario un menú vertical para que pueda
          elegir entre varias opciones, mediante las teclas del
          cursor o mediante un atajo (tecla caliente)
ENTRADAS: Las coordenadas de la esquina superior izquierda e inferior
          derecha de la zona asignada al menú, un vector de cadenas,
          terminado en NIL, en el que se señalan los atajos y la opción que
          hay que resaltar en primer lugar
SALIDAS:  Un número indicando la opción elegida, empezando a contar
          en 1, o 0 si no se elige ninguna
EJEMPLO:  Men_Vertical (1,10,8,20,{">Fichero","A>yuda",NIL},2)

FUNCION:  Min()
OBJETIVO: Calcular el mínimo de dos números
ENTRADAS: Los números
SALIDAS:  El menor de ellos
EJEMPLO:  Min (2,5)

FUNCION:  Pan_Borra()
OBJETIVO: Borrar una zona de la pantalla
ENTRADAS: El color y las coordenadas
SALIDAS:  Ninguna
EJEMPLO:  Pan_Borra ( ROJO,3,3,15,15 )

FUNCION:  Pan_BorraLinea()
OBJETIVO: Limpiar una línea determinada
ENTRADAS: La línea
SALIDAS:  Ninguna
EJEMPLO:  Pan_BorraLinea ( 0 )

FUNCION:  Pan_Caracter()
OBJETIVO: Escribir un carácter en la pantalla
ENTRADAS: El carácter
```

```
SALIDAS:  Ninguna
EJEMPLO:  Pan_Caracter ('A')

FUNCION:  Pan_Cierra()
OBJETIVO: Dejar la pantalla preparada para volver al sistema operativo
ENTRADAS: Ninguna
SALIDAS:  Ninguna
EJEMPLO:  Pan_Cierra ()

FUNCION:  Pan_Color()
OBJETIVO: Cambiar el color del papel y la tinta
ENTRADAS: El color del papel y de la tinta
SALIDAS:  Ninguna
EJEMPLO:  Pan_Color ( VERDE, NEGRO )

FUNCION:  Pan_Cursor()
OBJETIVO: Colocar el cursor en una posición determinada
ENTRADAS: La fila y la columna
SALIDAS:  Ninguna
EJEMPLO:  Pan_Cursor ( 0,0 )

FUNCION:  Pan_CursorVisible()
OBJETIVO: Poner y quitar el cursor
ENTRADAS: Lógica
SALIDAS:  Ninguna
EJEMPLO:  Pan_CursorVisible ( SI )

FUNCION:  Pan_Define()
OBJETIVO: Definir la pantalla principal del programa
ENTRADAS: El modo que se va a usar: texto o gráfico
NOTA:     Se usa una pantalla de 24 filas (de 0 a 23) y
          80 columnas (de 0 a 79)
EJEMPLO:  Pan_Define ( PAN_TEXTO )

FUNCION:  Pan_Entero()
OBJETIVO: Escribir un número entero en la pantalla en un ancho determinado
ENTRADAS: El número y el ancho
SALIDAS:  Ninguna
EJEMPLO:  Pan_Entero ( 12345,6 )

FUNCION:  Pan_Limpia()
OBJETIVO: Borrar la pantalla completa
ENTRADAS: Ninguna
SALIDAS:  Ninguna
EJEMPLO:  Pan_Limpia()

FUNCION:  Pan_Modo()
OBJETIVO: Decir el modo en que está definida la pantalla
ENTRADAS: Ninguna
SALIDAS:  PAN_TEXTO o PAN_GRAFICO
EJEMPLO:  Pan_Modo()


FUNCION:  Pan_Papel()
OBJETIVO: Cambiar el color de la pantalla
ENTRADAS: El nuevo color
SALIDAS:  Ninguna
EJEMPLO:  Pan_Papel ( VERDE )

FUNCION:  Pan_PonCar()
OBJETIVO: Escribir un carácter en cierta posición
ENTRADAS: La fila, la columna y el carácter
SALIDAS:  Ninguna
EJEMPLO:  Pan_PonCar (3,5,'A')

FUNCION:  Pan_PonEntero()
OBJETIVO: Escribir un entero en cierta posición
ENTRADAS: La fila, la columna, el entero y el ancho disponible
SALIDAS:  Ninguna
EJEMPLO:  Pan_PonEntero (3,5,12345,6)
```

```
FUNCION:  Pan_PonTexto()
OBJETIVO: Escribir un texto en cierta posición
ENTRADAS: La fila, la columna y el texto
SALIDAS:  Ninguna
EJEMPLO:  Pan_PonTexto (3,5,"Hola")

FUNCION:  Pan_Resalta()
OBJETIVO: Poner y quitar la situación de resaltado de caracteres
ENTRADAS: Lógica
SALIDAS:  Ninguna
EJEMPLO:  Pan_Resalta ( SI )

FUNCION:  Pan_Texto()
OBJETIVO: Escribir un texto en la pantalla
ENTRADAS: Una cadena
SALIDAS:  Ninguna
EJEMPLO:  Pan_Texto ("Hola, usuario")

FUNCION:  Pan_Tinta()
OBJETIVO: Cambiar el color con el que se escribe en la pantalla
ENTRADAS: El nuevo color
SALIDAS:  Ninguna
EJEMPLO:  Pan_Tinta ( ROJO )

FUNCION:  Prg_Presenta()
OBJETIVO: Presentar el programa
ENTRADAS: Nombre, versión, autor y fecha
SALIDAS:  Ninguna
EJEMPLO:  Prg_Presenta ( "Programa","0.0","Pedro Reina","1993" )

FUNCION:  Son_Apaga()
OBJETIVO: Apagar el sonido
ENTRADAS: Ninguna
SALIDAS:  La variable Son_Encendido se modifica y se devuelve
EJEMPLO:  Son_Apaga()

FUNCION:  Son_Bien()
OBJETIVO: Señalar brevemente que algo es correcto
ENTRADAS: Ninguna
SALIDAS:  El valor de Son_Encendido
EJEMPLO:  Son_Bien()

FUNCION:  Son_Cambia()
OBJETIVO: Cambiar el estado del sonido (Encendido / Apagado)
ENTRADAS: Ninguna
SALIDAS:  La variable Son_Encendido se modifica y se devuelve
EJEMPLO:  Son_Cambia()

FUNCION:  Son_Enciende()
OBJETIVO: Encender el sonido
ENTRADAS: Ninguna
SALIDAS:  La variable Son_Encendido se modifica y se devuelve
EJEMPLO:  Son_Enciende()

FUNCION:  Son_Error()
OBJETIVO: Señalar un error
ENTRADAS: Ninguna
SALIDAS:  El valor de Son_Encendido
EJEMPLO:  Son_Error()

FUNCION:  Son_MalaTecla()
OBJETIVO: Señalar que la tecla pulsada no se admite
ENTRADAS: Ninguna
SALIDAS:  El valor de Son_Encendido
EJEMPLO:  Son_MalaTecla()

FUNCION:  Tec_Disponible()
OBJETIVO: Decir si está disponible alguna tecla
ENTRADAS: Ninguna
```

```
      SALIDAS:  0 si no hay tecla disponible, 1 si la hay
      EJEMPLO:  Tec_Disponible()

      FUNCION:  Tec_FijadoMayus()
      OBJETIVO: Decir si está activado el sujeta-mayúsculas
      ENTRADAS: Ninguna, se lee del sistema
      SALIDAS:  Lógica
      EJEMPLO:  Tec_FijadoMayus()

      FUNCION:  Tec_Mayus()
      OBJETIVO: Convertir una tecla en mayúscula
      ENTRADAS: Una tecla
      SALIDAS:  La tecla convertida en mayúscula
      NOTA:     En esta versión no se tienen en cuenta caracteres acentuados
      EJEMPLO:  Tec_Mayus ( TEC_A )

      FUNCION:  Tec_Minus()
      OBJETIVO: Convertir una tecla en minúscula
      ENTRADAS: Una tecla
      SALIDAS:  La tecla convertida en minúscula
      NOTA:     En esta versión no se tienen en cuenta caracteres acentuados
      EJEMPLO:  Tec_Minus ( TEC_MAY_A )

      FUNCION:  Tec_Pertenece()
      OBJETIVO: Decidir si una tecla pertenece a cierto rango
      ENTRADAS: Una tecla y un vector con las teclas admitidas, terminado en NIL
      SALIDAS:  La tecla si pertenece o NIL si no pertenece
      EJEMPLO:  Tec_Pertenece ( TEC_F1, {TEC_ESC,NIL} )

      FUNCION:  Tec_Pulsada()
      OBJETIVO: Esperar a que el usuario pulse una tecla y devolver su código
      ENTRADAS: Ninguna
      SALIDAS:  El código asignado a la tecla pulsada
                La variable global Tec_Ultima_ queda modificada
      NOTA:     Deben usarse siempre los macros, puesto que los códigos
                dependen del compilador
      EJEMPLO:  Tec_Pulsada()

      FUNCION:  Tec_Ultima()
      OBJETIVO: Devolver la última tecla pulsada por el usuario
      ENTRADAS: Ninguna, se usa la variable global Tec_Ultima_
      SALIDAS:  El código asignado a Tec_Ultima_
      EJEMPLO:  Tec_Ultima()

      FUNCION:  Tec_Validada()
      OBJETIVO: Devolver una tecla de un determinado rango
      ENTRADAS: Un vector con las teclas admitidas, terminado en NIL
      SALIDAS:  La tecla pulsada
      EJEMPLO:  Tec_Validada ( {TEC_ESC,NIL} )

      FUNCION:  Tec_ValidadaRapido()
      OBJETIVO: Devolver una tecla de un determinado rango o NIL si no hay
                disponible ninguna tecla correcta
      ENTRADAS: Un vector con las teclas admitidas, terminado en NIL
      SALIDAS:  La tecla pulsada o NIL
      EJEMPLO:  Tec_ValidadaRapido ( {TEC_ESC,NIL} )

      FUNCION:  Tim_Ano()
      OBJETIVO: Decir el año en curso
      ENTRADAS: Ninguna
      SALIDAS:  El año en curso
      EJEMPLO:  Tim_Ano ()

      FUNCION:  Tim_Crono()
      OBJETIVO: Obtener una referencia temporal de precisión
      ENTRADAS: Ninguna, se consulta el reloj del sistema
      SALIDAS:  Un número real indicando el número de segundos desde la
                medianoche

      FUNCION:  Tim_Dia()
```

```
OBJETIVO: Decir el día en curso
ENTRADAS: Ninguna
SALIDAS:  El día en curso
EJEMPLO:  Tim_Dia ()

FUNCION:  Tim_Mes()
OBJETIVO: Decir el mes en curso
ENTRADAS: Ninguna
SALIDAS:  El mes en curso
EJEMPLO:  Tim_Mes ()

FUNCION:  Tim_Espera()
OBJETIVO: Suspender la ejecución del programa durante cierto tiempo
ENTRADAS: El tiempo en segundos
SALIDAS:  Ninguna

FUNCION:  Usr_Avisa()
OBJETIVO: Avisar al usuario de algo
ENTRADAS: El texto del aviso
SALIDAS:  Ninguna
EJEMPLO:  Usr_Avisa ( "Operación incorrecta" )

FUNCION:  Usr_BorraZona()
OBJETIVO: Borrar la zona de interacción con el usuario
ENTRADAS: El nuevo color de la zona
SALIDAS:  Ninguna
EJEMPLO:  Usr_BorraZona ( NEGRO )

FUNCION:  Usr_Consulta()
OBJETIVO: Preguntar algo al usuario
ENTRADAS: El texto de la pregunta
SALIDAS:  Lógica, según la contestación del usuario
EJEMPLO:  Usr_Consulta ( "¿Quieres seguir?" )

FUNCION:  Usr_Entero()
OBJETIVO: Devolver un número determinado por el usuario después de
          editar el que se le ofrece
ENTRADAS: El número que hay que editar, el ancho asignado, los valores
          mínimo y máximo admitidos, las coordenadas donde se edita y
          los colores
SALIDAS:  El entero editado
EJEMPLO:  Usr_Entero (100,4,-900,2000,1,1,BLANCO,NEGRO)

FUNCION:  Usr_Indica()
OBJETIVO: Mandar un mensaje al usuario
ENTRADAS: Dos cadenas
SALIDAS:  Ninguna
EJEMPLO:  Usr_Indica ( "Elige una opción", "Pulsa ENTER" )

FUNCION:  Usr_Informa()
OBJETIVO: Mandar un mensaje al usuario
ENTRADAS: El texto del mensaje
SALIDAS:  Ninguna
EJEMPLO:  Usr_Informa ( "Calculando" )

FUNCION:  Usr_PulsaUnaTecla()
OBJETIVO: Mandar un mensaje al usuario y esperar que pulse una tecla
ENTRADAS: El texto del aviso
SALIDAS:  Ninguna
EJEMPLO:  Usr_PulsaUnaTecla ( "Operación terminada." )

FUNCION:  Usr_Texto()
OBJETIVO: Devolver una cadena determinada por el usuario después de
          editar la que se le ofrece
ENTRADAS: La cadena que hay que editar, el ancho asignado,
          las coordenadas donde se edita y los colores
SALIDAS:  Una cadena
NOTA:     La cadena devuelta hay que destruirla cuando no sea necesaria
EJEMPLO:  Usr_Texto ("Artesonado",20,1,1,BLANCO,NEGRO)
```

```
FUNCION:  Zon_AltoFisico()
OBJETIVO: Decir la altura en pixels de una zona
ENTRADAS: La zona
SALIDAS:  Un contador que indica la altura física
EJEMPLO:  Zon_AltoFisico ( Imagen )

FUNCION:  Zon_AnchoEnOxel()
OBJETIVO: Decir la anchura en oxels de una zona
ENTRADAS: La zona
SALIDAS:  Un contador que indica la anchura en oxels
EJEMPLO:  Zon_AnchoEnOxel ( Imagen )

FUNCION:  Zon_AnchoFisico()
OBJETIVO: Decir la anchura en pixels de una zona
ENTRADAS: La zona
SALIDAS:  Un contador que indica la anchura física
EJEMPLO:  Zon_AnchoFisico ( Imagen )

FUNCION:  Zon_Borra()
OBJETIVO: Borrar una zona con un color
ENTRADAS: La zona y un contador que indica el color
SALIDAS:  Ninguna
EJEMPLO:  Zon_Borra ( Imagen, NEGRO )

FUNCION:  Zon_Crea()
OBJETIVO: Crear una zona
ENTRADAS: La fila superior, la columna izquierda, la fila inferior
          y la columna derecha, todo en caracteres
SALIDAS:  Una zona o NIL si no se puede crear
EJEMPLO:  Zon_Crea (3,4,14,70)

FUNCION:  Zon_Destruye()
OBJETIVO: Eliminar una zona, liberando memoria
ENTRADAS: La zona
SALIDAS:  Ninguna
EJEMPLO:  Zon_Destruye ( Imagen )

FUNCION:  Zon_Oxel()
OBJETIVO: Dibujar un oxel de una zona de un color
ENTRADAS: La zona, los valores "x" e "y" del oxel, la máscara de dibujo
          y el color
SALIDAS:  Lógica, indicando si se ha podido dibujar
NOTAS:    1. Los valores "x" e "y" se cuentan a partir de 0 desde la
             izquierda y desde arriba respectivamente
EJEMPLO:  Zon_Oxel ( Imagen, 6, 7, 0xFF, ROJO )

FUNCION:  Zon_OxelSeguro()
OBJETIVO: Dibujar un oxel que pertenezca a una zona de un color
ENTRADAS: La zona, los valores "x" e "y" del oxel, la máscara de dibujo
          y el color
SALIDAS:  SI, ya que debe poder ejecutarse
NOTAS:    1. Los valores "x" e "y" se cuentan a partir de 0 desde la
             izquierda y desde arriba respectivamente
          2. Para ejecutar esta función el oxel debe pertenecer a la zona
EJEMPLO:  Zon_OxelSeguro ( Imagen, 0, 5, 0xFF, ROJO )

FUNCION:  Zon_Pixel()
OBJETIVO: Dibujar un pixel de una zona de un color
ENTRADAS: La zona, los valores "x" e "y" del pixel y el color
SALIDAS:  Lógica, indicando si se ha podido dibujar
NOTA:     Los valores "x" e "y" se cuentan a partir de 0 desde la
          izquierda y desde arriba respectivamente
EJEMPLO:  Zon_Pixel ( Imagen, 3, 8, ROJO )

FUNCION:  Zon_PixelSeguro()
OBJETIVO: Dibujar un pixel que pertenezca a una zona de un color
ENTRADAS: La zona, los valores "x" e "y" del pixel y el color
SALIDAS:  SI, ya que debe poder ejecutarse
NOTAS:    1. Los valores "x" e "y" se cuentan a partir de 0 desde la
             izquierda y desde arriba respectivamente
```

2. Para ejecutar esta función el pixel debe pertenecer a la zona
EJEMPLO: Zon_PixelSeguro (Imagen, 3, 8, ROJO)

Cómo hacer cambios

El código está dividido en la mayor cantidad posible de ficheros para facilitar la realización de cambios.

Cada objeto (y las definiciones generales) tiene un fichero de cabecera.

Los nombres de los ficheros *.c reflejan el objeto al que se refieren (las tres primeras letras) y la función más importante que definen.

Es muy importante validar las funciones que se cambien o añadan, y para ello se dispone de ficheros de validación, que tienen como nombre Vobj.c, donde obj son las tres letras que identifican cada objeto. El fichero de definiciones generales se identifica con "gen".

Cuando se realiza un cambio en un objeto, hay que comprobar si ese cambio afecta a otros objetos que dependan de él, por lo que es muy aconsejable seguir cierto orden al realizar cambios. Hay que ir de los más básicos a los más complejos.

Esta es la relación de interdependencias estre objetos:

Objetos necesarios	Objeto	Objetos que dependen de éste
	Pan	Cdr Usr Men Prg Zon
Pan	Zon	
	Azr	
	Tim	Bdt
	Cad	Usr Fch Men Prg
	Lis	
	Son	Tec Usr
Son	Tec	Usr Men
Pan	Cdr	Usr
Pan Cad Son Tec Cdr	Usr	Fch Men
Usr Cad	Fch	Cnf Bdt
Cad Pan Usr Tec	Men	
Pan Cad	Prg	
Fch	Cnf	
Fch Usr Tim Cad	Bdt	

General (Gen)

Este objeto consta de los siguientes ficheros:

General.h

Pantalla (Pan)

Este objeto consta de los siguientes ficheros:

Pantalla.def
Pantalla.h
PanBloq.c
PanDef.c
PanResal.c

Zona (Zon)

Este objeto consta de los siguientes ficheros:

Zona.h
ZonCrea.c
ZonOxFis.c
ZonPixFi.c

Azar (Azr)

Este objeto consta de los siguientes ficheros:

Azar.h

Tiempo (Tim)

Este objeto consta de los siguientes ficheros:

Tiempo.h
TimAno.c
TimDia.c
TimMes.c

Cadena (Cad)

Este objeto consta de los siguientes ficheros:

Cadena.h
CadCamb.c
CadCarPe.c
CadCrea.c
CadEnt.c
CadMueve.c
CadPrimU.c
CadQuita.c
CadSub.c
CadTrozo.c
CadUne.c

Lista (Lis)

Este objeto consta de los siguientes ficheros:

Lista.h
LisAgFin.c
LisAgreg.c
LisCrea.c
LisDestr.c
LisTotal.c

Sonido (Son)

Este objeto consta de los siguientes ficheros:

Sonido.def
Sonido.h
SonEjec.c

Tecla (Tec)

Este objeto consta de los siguientes ficheros:

Tecla.def
Tecla.h
TecFijMa.c
TecPerte.c
TecPulsa.c
TecValid.c
TecVaRap.c

Cuadro (Cdr)

Este objeto consta de los siguientes ficheros:

Cuadro.def
Cuadro.h
CdrDibuj.c
CdrLinea.c

Usuario (Usr)

Este objeto consta de los siguientes ficheros:

Usuario.h
UsrAvisa.c
UsrBorrZ.c
UsrConsu.c
UsrEnt.c
UsrIndic.c
UsrInfor.c
UsrInMay.c
UsrInMod.c
UsrPulsa.c
UsrTexto.c

Fichero (Fch)

Este objeto consta de los siguientes ficheros:

Fichero.h
FchAbre.c
FchAbreA.c
FchAbreG.c
FchAbreL.c
FchBorra.c
FchEscrL.c
FchEscrO.c
FchExist.c
FchLeeL.c
FchLeeO.c
FchNombr.c

Menú (Men)

Este objeto consta de los siguientes ficheros:

Menu.h
MenEscr.c
MenExam.c
MenHoriz.c

MenTotal.c
MenVert.c

Programa (Prg)

Este objeto consta de los siguientes ficheros:

Programa.h
PrgPres.c

Configuración (Cnf)

Este objeto consta de los siguientes ficheros:

Config.h
CnfLee.c

Base de datos (Bdt)

Este objeto consta de los siguientes ficheros:

Basedato.h
BdtAbre.c
BdtAgrRe.c
BdtCamNo.c
BdtCamNu.c
BdtCrea.c
BdtCreOb.c
BdtDecCa.c
BdtEscRe.c
BdtLeeRe.c
BdtLonCa.c
BdtMarRe.c
BdtNomCa.c
BdtNumCa.c
BdtPCaNo.c
BdtPCaNu.c
BdtTipCa.c

Una vez realizados los cambios, hay que volver a formar los ficheros Olimpo.h y Olimpo.lib.

Para crear Olimpo.h he escrito el programa UneH, que lee todos los ficheros de cabecera, les quita la información que no es necesaria, y escribe Olimpo.h.

Para crear Olimpo.lib se dispone del fichero CreaLib.bat. Para el QL es necesario examinar el orden en que hay que montar los distintos _o.

Hay una serie de makefiles, llamados Vobj.mak (donde obj se sustituye por el identificador de cada objeto) que sirven para tener al día los ficheros de validación.

En el QL he creado un programa SuperBasic de soporte del desarrollo. Precisamente se llama Soporte_bas. Cuando se arranca, presenta una pantalla de ayuda que explica sus posibilidades.

Historia de las versiones
=========================

La idea de realizar Olimpo proviene de intentar resolver varios problemas:

* Llevaba mucho tiempo programando y me encontraba que estaba escribiendo las mismas rutinas una y otra vez. Había que encontrar el método para utilizar las rutinas ya escritas en programas nuevos.

* Tengo tres sistemas operativos y me gustaría que mis programas funcionaran, sin cambiar el código, en los tres. El lenguaje que me permite hacer esto es el C.

* Me gustaría escribir métodos generales de resolución de problemas para estudiar inteligencia artificial, pero para poder presentar un programa debe tener un mínimo interfaz; por tanto, hay que empezar por programar las cosas básicas y sobre ellas ir montando métodos mas complejos.

Cuando se realiza cualquier sistema de programación es muy aconsejable que las nuevas versiones no cambien los protocolos de las funciones, para no tener que retocar el código de un programa escrito sobre el sistema al cambiar de versión. Desafortunadamente, yo no tengo claras ciertas cosas y además estoy escribiendo el sistema partiendo de algo sencillo y añadiéndole utilidades poco a poco; por tanto, será inevitable que algunas funciones cambien de protocolo conforme aumente el número de versión de Olimpo. Por supuesto, iré documentando los cambios.

Versión 0.0

Es la versión en la que se comienzan a construir las bases del sistema.

Se dispone de un conjunto de definiciones generales y de 12 objetos:

General, Pantalla, Azar, Tiempo, Cadena, Sonido, Tecla, Usuario, Fichero, Menú, Programa, Cuadro y Configuración.

El sistema funciona por medio del fichero Olimpo.h, cuyo único cometido es llamar a los distintos .h de los objetos. Por tanto, el tiempo de compilación es muy elevado.

Versión 0.1

Reúno todos los .h para formar el fichero Olimpo.h.

Retoco el objeto Fichero, añadiendo Fch_Borra() y un manejador de interrupciones en el PC.

Versión 1.0

Separo lo más posible el código en varios ficheros, que compilo por separado y creo un fichero de librería. Además, escribo un programa que permite unir todos los .h del sistema quitando todas las redundancias y formando Olimpo.h, único fichero de cabecera que debe utilizar el programador usuario del sistema.

Reescribo completamente el objeto Cadena, utilizando asignación dinámica.

El objeto Pantalla en el PC ahora admite trabajar en modo gráfico, lo que abre las posibilidades de crear y añadir objetos gráficos a Olimpo en un futuro.

Reescribo completamente los programas de validación y creo métodos para el mantenimiento de los ficheros.

Todas estan funciones han sido retocadas de modo que hay que revisar el código que las utilice:

Pan_Define(), Pan_Entero(), Usr_Consulta(), Usr_Texto(), Fch_Abre(), Fch_AbreLeer(), Fch_AbreGrabar(), Fch_LeeLinea(), Fch_EscribeLinea(), Fch_Nombre() y todas las del objeto Cadena.

Versión 1.1

Añado los objetos Zona, Lista y Base de datos.

Añado las funciones Pan_Modo(), Cad_Igual(), Tim_Ano, Tim_Mes(), Tim_Dia(), Fch_Coloca(), Fch_ColocaFinal() y Fch_AbreActualizar().

Añado el tipo octeto.

Preparo makefiles para la versión QL. Mejoro el programa de soporte.

Errores
=======

Todo el software que conozco tiene errores. Parece que es algo consustancial con el ser humano: equivocarse. Lo primero que hay que hacer para arreglar un error es conocerlo. Agredezco la ayuda de todos los usuarios de Olimpo que encuentran errores en él.

Versión 0.0

Usr_Consulta() -> En la versión PC el cursor a veces aparece y a veces no. No he encontrado el patrón del error; sospecho que es problema de la BIOS.

Cad_ConvierteEntero() -> tiene distinta implementación en QL que en PC y no estoy seguro de que funcione bien para enteros muy grandes.

Objeto Fichero -> en la versión PC hay que añadir un manejo de interrupciones, usando harderr(), ya que las interrupciones de hardware, como intentar escribir un un disquete protegido de escritura, las trata el sistema y rompe el diseño de la pantalla. Añadido en la versión 0.1.

Versión 1.0

Usr_Consulta() -> aun habiendo cambiado la función Pan_CursorVisible(), el error presente en la versión 0.0 sigue apareciendo.

Usr_Texto() -> no funciona la tecla cursor derecha cuando el ancho asignado es igual a la longitud de la cadena. Arreglado en la versión 1.1.

No se puede distribuir un programa en varios ficheros porque al incluir más de una vez Olimpo.h se producen errores. Arreglado en la versión 1.1.

CnfLee() -> Deja memoria sin devolver. Arreglado en la versión 1.1.

Men_Horizontal(), Men_Vertical() -> producen errores cuando se mandan más opciones de las que pueden manejar. Arreglado en la versión 1.1.

Fch_LeeOcteto() -> no da error cuando no puede leer tantos octetos como se le pide. Arreglado en la versión 1.1.

Versión 1.1

Objeto Zona -> en el QL hay una pequeña discrepancia entre cada zona y los pixels más a la derecha que se pueden representar de esa zona.

Futuras mejoras

Olimpo está en constante desarrollo, ya que intento que lo que voy aprendiendo y necesitando en informática se vaya incorporando al sistema. Es el mejor método que conozco para aprovechar al máximo el tiempo: reutilizar código.

Estas son algunas de las ideas que pretendo ir incorporando a cada objeto:

Cuadro

Deseo escribir la función Cdr_Construye(), que devuelva cadenas rellenas con cuadros, para poder mandarlos a la impresora.

Impresora

Es un objeto nuevo, que deberá encargarse de la gestión de impresoras.

Zona

El objeto zona no será realmente útil hasta que no cuente con la posibilidad de realizar gráficos que sean independientes de la resolución del monitor.

El primer paso es la función Zon_Escala(), que permitirá definir la escala que se desea usar; luego deben llegar funciones para dibujar las distintas figuras necesarias.

Base de datos

Hay que escribir funciones que permitar leer y escribir campos en otros formatos más cómodos: lógico, número, etc., y no sólo como cadenas, como ocurre ahora.

Independientemente, deseo escribir código que maneje ficheros índice en formato NTX, compatible con Clipper.

Fichero

Deseo escribir una rutina que permita elegir un fichero desde una lista de ficheros disponibles.

Agradecimientos y bibliografía

Los conceptos de la programación orientada el objeto que constituyen la columna vertebral de este sistema las aprendí de la empresa "EQ, sistemas inteligentes", que es una empresa especializada en inteligencia artificial, software basado en el conocimiento y pedagogía de estos conceptos. Para más información:

EQ sistemas inteligentes
Av. General Perón 19, bajo C
28020 Madrid
Teléfonos 556 53 09 y 555 10 19
Fax 556 53 09

Todo el código de Olimpo es original. Aunque he recogido ideas en varios sitios,

algunas cruciales y muy bien expuestas, el código final ha salido de mis dedos, no de "cortar y pegar".

Los manuales de los compiladores de C que utilizo para desarrollar Olimpo son muy importantes; sin ellos no podría desarrollar nada. En el PC utilizo un compilador de Borland. La delegación española está en:

Borland Ibérica S.A.
Edificio Borland
c/ Azalea, 1
28100 Soto de la Moraleja
Madrid

Para el QL uso el que sin ninguna duda es el mejor compilador de C disponible, C68. Este compilador es de dominio público y no sería posible contar con él sin el esfuerzo desinteresado de muchas personas. C68 se puede conseguir en cualquier distribuidor de software de dominio público para QL. El actual coordinador del equipo de desarrollo es:

Dave Walker
22 Kimptons Mead
Potters Bar
Herts, EN6 3HZ
United Kingdom

Como complemento al manual de C68 es muy conveniente usar algún libro sobre el QDOS. El que suelo manejar es:

Título: QL Programación Avanzada
Autor: Adrian Dickens
Editorial: ra-ma

Las funciones de manejo de la tarjeta gráfica VGA del PC las he aprendido en:

Título: Graphics Programming in C
Autor: Roger T. Stevens
Editorial: Prentice Hall

Para desarrollar los objetos Tecla y Menú me ha sido de mucha utilidad el conocimiento del lenguaje Clipper en su versión 5.01, que en aquel momento pertenecía a Nantucket. La versión actual es otra, y el producto es de Computer Associates.

Asímismo me ha sido útil en el desarrollo del objeto Menú el libro:

Título: Designing Screen Interfaces in C
Autor: James Pinson
Editorial: Yourdon Press

Para el desarrollo del objeto Base de datos me he basado exclusivamente en la documentación que acompaña al producto "SoftC Database Library", desarrollado por la empresa

SoftC Limited
16820 Third Street Northeast
Anoka, MN 55304-4703
U.S.A.

En el momento de escribir estas notas quien distribuye la librería es:

Greenleaf Software
16479 Dallas Parkway

Suite 570
Dallas, TX 75248
U.S.A.

Programa de demostración

Culquier sistema de ayuda a la programación está pensado para desarrollar programas, de modo que el mejor modo de comprender cómo se usa Olimpo es verlo realizar algo concreto. Por ese motivo, he desarrollado en programa de demostración. No pretende ser exhaustivo, sino un punto de arranque para que el usuario de Olimpo pueda empezar lo antes posible a desarrollar su propio código.

Si estás leyendo la versión en disquete de esta documentación, consulta el fichero Demo.c; en el caso de que estés leyendo la versión en papel, más tradicional y creo que también más cómoda, puedes ver el fichero impreso al final de estas notas.

MERINO TIL v2.0

Poco hay que contar. La v2.0 ha sido actualizada con la nueva libreria OLIMPO v1.1 de Pedro Reina. Y la novedad más importante es la compatibilidad con los ficheros dBase III+, pues gracias a las rutinas escritas en 'C' por Pedro Reina, ahora podemos trabajar con ficheros DBF de dBase III+ directamente.

Los socios que sean Usuarios Registrados de MERINO TIL pueden actualizar su código fuente enviando un disco formateado. Su disco será retornado grabado con la última versión junto con el nuevo QLíper.

Salvador Merino, 25/1/1994

VER PANTALLA QL

Acababa de escribir Olimpo 1.1 y ya estaba deseando usarlo. Había escrito código para manejar la pantalla gráfica VGA del PC y la del QL de una manera muy parecida. No sólo es posible manejar cada pixel individualmente, sino que cada grupo de 8 pixels también se puede manejar.

Me explico: el QL en modo 4 maneja, como ya sabéis, un grupo de 8 pixels mediante 2 octetos (a partir de la dirección 131072): el primero sirve para saber qué pixels van en verde y el segundo para los que van en rojo; los bits que están activos en ambos octetos señalan los bits que están en blanco. Todo esto ya lo sabíais. Pero es que la VGA funciona de una manera parecida. Es posible cambiar de golpe el color de 8 pixels mediante una sólo escritura.

Por tanto, he definido el "oxel" (abreviatura de octeto y pixel) como el conjunto de 8 pixels que maneja el hardware globalmente.

El siguiente paso fue escribir una función que maneje de manera uniforme los oxels del PC y del QL para que el programador pueda obviar sobre qué ordenador está programando.

Una vez que terminé de escribir esta función, estaba chupado escribir un programa que permitiera ver una pantalla QL en un PC equipado con VGA. El código de ese programa lo tenéis en el fichero VerPanQL_c. Para recompilarlo necesitáis Olimpo 1.1 y Turbo C.

Para usar el programa, basta llevarse el fichero VerPanQL.exe a un PC usando cualquier programa de transferencia de ficheros. A ver si os gusta.

Un último detalle: si véis con este programa cualquier pantalla, estaréis viendo también cuánto espacio ocupa en una pantalla VGA la tarjeta QXL, es el mismo, ya que la QXL sólo utiliza 512x256 pixels de los 640x480 que tiene la VGA.

Pedro Reina, J.13.1.1994